



UNIVERSITY OF SOUTHERN DENMARK

DM852

---

## Heuristics & Approximations Algorithms

---

*Submitted To:*

Marco Chiarandini  
Lene Monrad Favrholt  
IMADA  
Mathematics & Computer  
Science Department

*Submitted By :*

Alexander Lerche Falk  
Narongrit Unwerawattana  
Spring - Master of  
Computer Science

## Contents

## 1 Introduction

This project shows metaheuristic algorithms, resolving the Capacitated Vehicle Routing Problem (CVRP). The difference between metaheuristics and heuristics is in the solution part. For heuristics, you are trying your best to find a solution, even though it is not optimal. The algorithm is adapted to the problem in such a greedy approach, it can get stuck in a local optimum. This is fine since it is the idea of heuristics: "just solve the problem".

Metaheuristics are less greedy and tends to be more problem independent. They accept temporary solutions and allow "bad" steps as an attempt to get a better solution. You have a local- and global optimum to keep track of the best solution found. You can say metaheuristics are exploiting heuristics to avoid getting trapped. In this metaheuristics implementation project, we have chosen to implement two algorithms for CVRP: Simulated Annealing (SA) and Ant Colony Optimization (ACO). The SA algorithm is the one we have uploaded for electronic submission at <http://valkyria.imada.sdu.dk/D0App/>. The ACO algorithm is implemented but does not perform well. We will compare the algorithms with our previous heuristic / local search project and lastly, show our results of computation for our two algorithms.

— Alexander & Narongrit

## 2 Simulated Annealing

The Simulated Annealing (SA) algorithm is inspired from the annealing process in metal, where you are altering the physical state of the metal by heating and cooling it. The inspiration can be used in computer science as well. We can use the algorithm on CVRP by starting off by generating a solution to the problem and not "care" about the initial solutions. The better steps we are taking, and better solutions we are finding, the more careful we are going to be in finding a solution. In the beginning we allow random and bad steps to be performed, while later, we make better calculations.

In our algorithm we have ... KIE DO SOMETHING HERE

---

**Algorithm 1** Metaheuristic - Simulated Annealing

---

### 3 Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm comes from the evolutionary algorithms, where computer science meets nature. In this algorithm, we have led us to be inspired by how ants find food in the nature. They are starting by spreading out in some area, randomly. When they seem to find trails, which could indicate to be good trails to find food, they leave pheromone behind them. The pheromone is used by other ants to determine their probability of taking a trail. If they are standing in a cross-section and they have to choose, they are taking the path with the highest pheromone. At some point in their exploration to find the best trail to obtain food, all the ants are using the same trails to get food and get back safe home.

We can apply the logic of the ants in the CVRP as well. By letting the instance the space of which the requests are "plotted" be the area to find a solution, we can start by sending one "ant" out to a random point. This is going to be our current point. From this point, we are going to calculate every probability of moving to the next point the one with the highest "score". We can calculate the probability of moving by the doing the following:

First we establish the initial pheromone levels from all the points to every counter other point:

This is stored in matrix:  $M = \begin{bmatrix} 0 & 0.50 & 0.20 \\ 0.125 & 0 & 0.60 \\ 1.20 & 0.355 & 0 \end{bmatrix}$

We also keep track of the points being visited, so we do not keep visiting the same nodes. Then we start calculating a score for each point from the current point. We do this by the formular:

$$score = \alpha^{\frac{1.0}{distance(i,j)}} \beta \quad (1)$$

The  $\alpha$  value defines the importance of the pheromone level, where the  $\beta$  value defines the importance of the distance between two points. After finding the score for each point from the current point, we can calculate the probability by dividing the score by the summation of all scores:

$$prob = \frac{score}{\sum_{i=1}^n score_i} \quad (2)$$

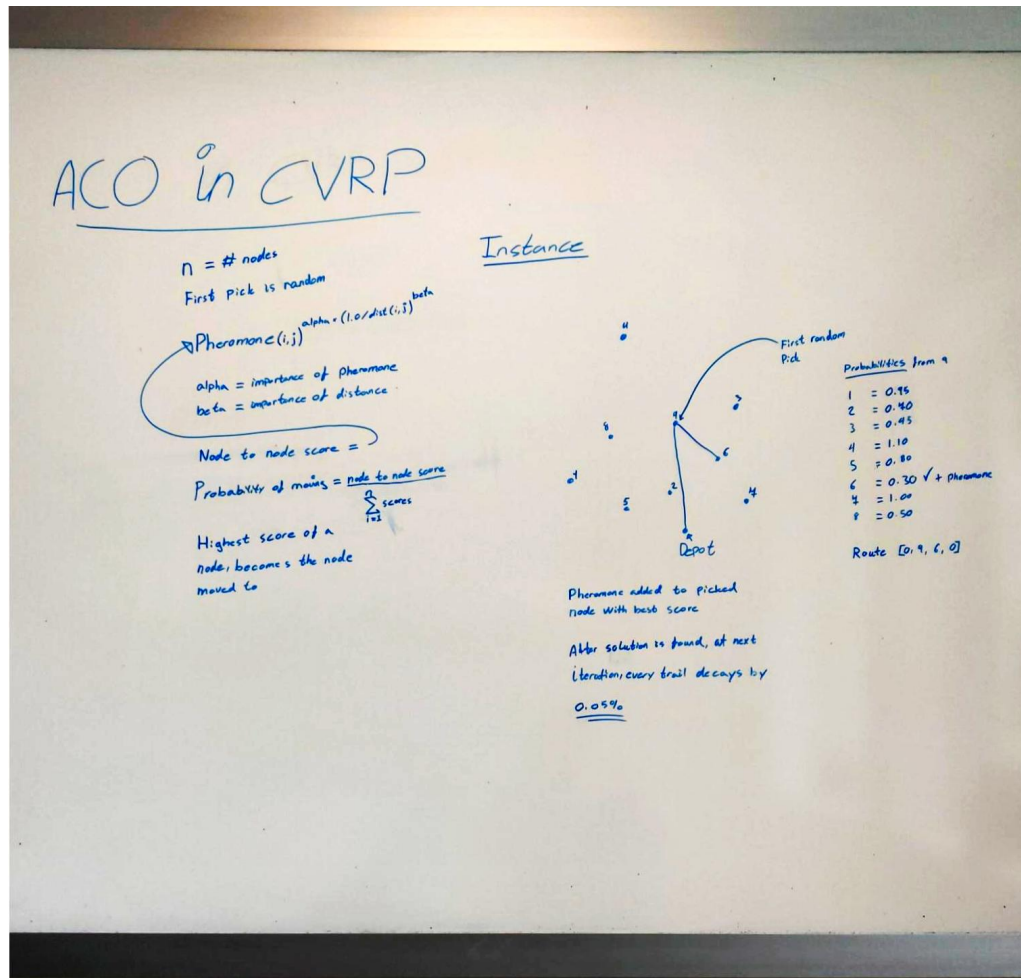
Where  $\mathbf{n}$  is equivalent to the number of points. We can now append additional pheromone to the trail from the current point to the next point, by dividing:

$$newpheromone = \frac{1}{distance(i, j)} \quad (3)$$

Now we can continue with our usual CVRP properties, where we ensure the capacity of the vehicle is not breached. We continue finding points with the above algorithm. After the algorithm has found a solution, we are going to run it again, with the same pheromone matrix, but with updated values, where every pheromone in the trails evaporates a small percentage before each iteration. A timer is being set, which controls for how long the algorithm runs.

In Figure ?? a drawing has been created to visualize the idea of the algorithm:

Figure 1: Illustration of the ACO algorithm, showing how it can be used, and how it process next moves



The pseudocode for the algorithm is described as followed:

Giving us the next bloxpawdaw

4

## BOXPLOT FIGURES

---

**Algorithm 2** Metaheuristic - Ant Colony Optimization

---

```

1: function ALGORITHM
2:   capacity  $\leftarrow$  0
3:   alpha  $\leftarrow$  A random number
4:   beta  $\leftarrow$  A random number greater than alpha
5:   evaporation  $\leftarrow$  Some small number
6:   pheromone_matrix  $\leftarrow$  Contains every pheromone for each pair
7:   sum_of_all_pheromone  $\leftarrow$  The summation of the pheromone matrix Setup
   Initial Pheromone
8:   for first_point  $\leftarrow$  0 to length of data do
9:     for second_point  $\leftarrow$  0 to length of data do
10:      if first_point  $\neq$  second_point then                                pheromone(first, second)  $\leftarrow$  1/dis
11:      if
12:        then
13:
14:

```

---

## Appendices